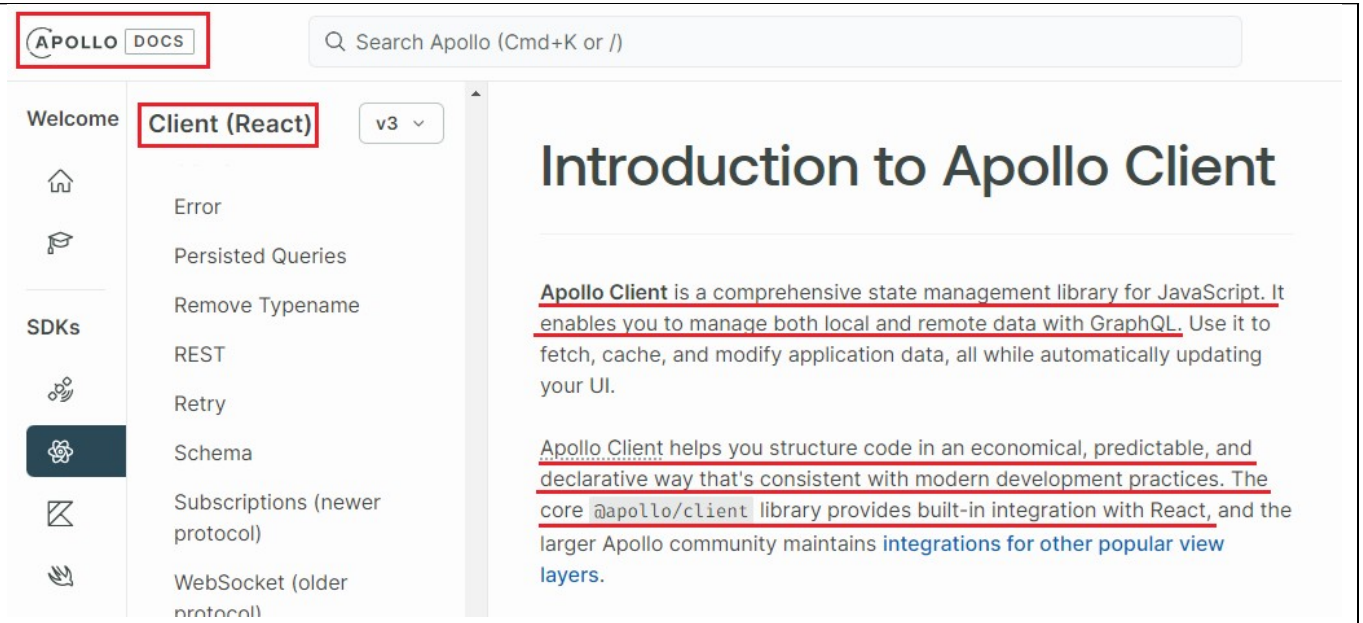
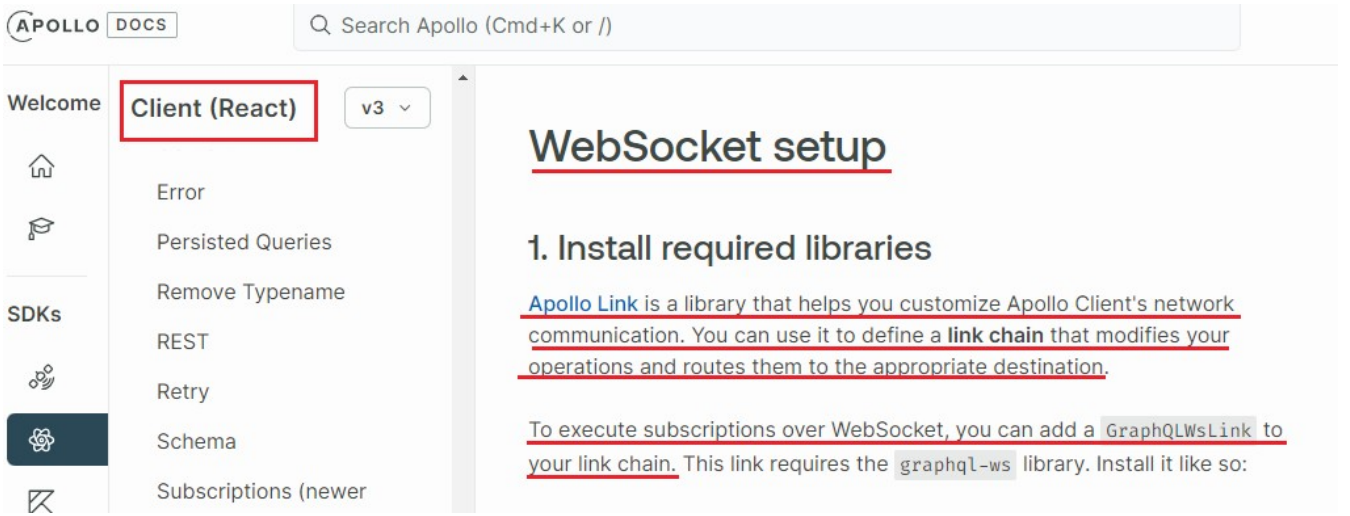
Exhibit 2

Charted Claim:

Method Claim: 21

US7756983	Apollo Client (“Accused Product”)
21. A method of computer network node communication comprising: initially establishing a first asymmetric hypertext transfer protocol (HTTP) transactional session to establish an original network connection, said session being uniquely identifiable, between a first node and a second node, said first node enacting a first transactional role of a client and said second node enacting a second transactional role of a server, said roles	The accused product, compliant with RFC 6455, practices a method of computer network node communication comprising: initially establishing a first asymmetric hypertext transfer protocol (HTTP) transactional session (e.g., HTTP session) to establish an original network connection (e.g., TCP connection), said session being uniquely identifiable (e.g., Request-URI), between a first node (e.g., client) and a second node (e.g., server), said first node enacting a first transactional role of a client and said second node enacting a second transactional role of a server, said roles comprising either of an HTTP server that relays data and an HTTP client that initiates requests. As shown below, WebSocket protocol, which is defined under IETF RFC 6455, enables two-way communication between a web browser and a server. The WebSocket protocol provides a bidirectional, full-duplex communications channel that operates over HTTP through a single TCP/IP socket connection. The WebSocket protocol uses HTTP as the initial transport mechanism and uses the TCP connection. This connection is used for sending messages between a client and a server. WebSocket starts with a standard HTTP requests and response session. Within the request-response chain, the client asks to open a WebSocket connection, and then the server responds. Thus, the first and the second nodes are engaged in an asymmetric hypertext transfer protocol (HTTP) transactional session with an original network connection. The first node acts as an HTTP client that initiates requests and the second node acts as an HTTP server that relays data. Also, in the request-response chain, a Request-URI is used to identify the endpoint of the WebSocket connection. Thus, it can be said that the session is uniquely identifiable.

comprising either of an HTTP server that relays data and an HTTP client that initiates requests;	 https://www.apollographql.com/docs/react/  https://www.apollographql.com/docs/react/data/subscriptions#websocket-setup

Internet Engineering Task Force (IETF)
Request for Comments: 6455
Category: Standards Track
ISSN: 2070-1721

I. Fette
Google, Inc.
A. Melnikov
Isode Ltd.

December 2011

The WebSocket Protocol

Abstract

The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is the origin-based security model commonly used by web browsers. The protocol consists of an opening handshake followed by basic message framing, layered over TCP. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g., using XMLHttpRequest or <iframe>s and long polling).

https://www.rfc-editor.org/rfc/rfc6455

A simpler solution would be to use a single TCP connection for traffic in both directions. This is what the WebSocket Protocol provides. Combined with the WebSocket API [WSAPI], it provides an alternative to HTTP polling for two-way communication from a web page to a remote server.

https://www.rfc-editor.org/rfc/rfc6455

The WebSocket Protocol is designed to supersede existing bidirectional communication technologies that use HTTP as a transport layer to benefit from existing infrastructure (proxies, filtering, authentication). Such technologies were implemented as trade-offs between efficiency and reliability because HTTP was not initially meant to be used for bidirectional communication (see [RFC6202] for further discussion). The WebSocket Protocol attempts to address the goals of existing bidirectional HTTP technologies in the context of the existing HTTP infrastructure; as such, it is designed to work over HTTP ports 80 and 443 as well as to support HTTP proxies and intermediaries, even if this implies some complexity specific to the current environment. However, the design does not limit WebSocket to HTTP, and future implementations could use a simpler handshake over a

https://www.rfc-editor.org/rfc/rfc6455

How Do Websockets Work?

A WebSocket is a persistent connection between a client and server. WebSockets provide a bidirectional, full-duplex communications channel that operates over HTTP through a single TCP/IP socket connection. At its core, the WebSocket protocol facilitates message passing between a client and server. This article provides an introduction to the WebSocket protocol, including what problem WebSockets solve, and an overview of how WebSockets are described at the protocol level.

https://sookocheff.com/post/networking/how-do-websockets-work/

WebSockets, on the other hand, allow for sending message-based data, similar to UDP, but with the reliability of TCP. WebSocket uses HTTP as the initial transport mechanism, but keeps the TCP connection alive after the HTTP response is received so that it can be used for sending messages between client and server. WebSockets allow us to build "real-time" applications without the use of long-polling.

https://sookocheff.com/post/networking/how-do-websockets-work/

	WebSockets begin life as a standard HTTP request and response. Within that request response chain, the client asks to open a WebSocket connection, and the server responds (if its able to). If this initial handshake is successful, the client and server have agreed to use the existing TCP/IP connection that was established for the HTTP request as a WebSocket connection. Data can now flow over this connection using a basic framed message protocol. Once both parties acknowledge that the WebSocket connection should be closed, the TCP connection is torn down. https://sookocheff.com/post/networking/how-do-websockets-work/

	1.3. Opening Handshake _This section is non-normative._ The opening handshake is intended to be compatible with HTTP-based server-side software and intermediaries, so that a single port can be used by both HTTP clients talking to that server and WebSocket clients talking to that server. To this end, the WebSocket client's handshake is an HTTP Upgrade request: GET /chat HTTP/1.1 Host: server.example.com Upgrade: websocket Connection: Upgrade Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== Origin: http://example.com Sec-WebSocket-Protocol: chat, superchat Sec-WebSocket-Version: 13 In compliance with [RFC2616], header fields in the handshake may be sent by the client in any order, so the order in which different header fields are received is not significant. The "Request-URI" of the GET method [RFC2616] is used to identify the endpoint of the WebSocket connection, both to allow multiple domains to be served from one IP address and to allow multiple WebSocket endpoints to be served by a single server. The client includes the hostname in the |Host| header field of its handshake as per [RFC2616], so that both the client and the server can verify that they agree on which host is in use. https://www.rfc-editor.org/rfc/rfc6455
terminating said first asymmetric HTTP transactional session while maintaining an	The accused product practices terminating said first asymmetric HTTP transactional session (e.g., HTTP session) while maintaining an underlying network connection (e.g., TCP connection). The WebSocket protocol starts with standard HTTP requests and responses. Within

underlying network connection;	the request-response chain, the client asks to open a WebSocket connection, and the server responds. If this initial handshake is successful, the client and server agree to use the existing TCP/IP connection that was established for the HTTP request as a WebSocket connection. Here, the TCP/IP connection is the original network connection aka the underlying network connection. WebSocket communication is established by upgrading an HTTP request/response. Hence, the asymmetric HTTP transactional session is terminated when WebSocket connection is established, and the existing TCP/IP connection is maintained.

	1.2. Protocol Overview _This section is non-normative._ The protocol has two parts: a handshake and the data transfer. The handshake from the client looks as follows: GET /chat HTTP/1.1 Host: server.example.com Upgrade: websocket Connection: Upgrade Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== Origin: http://example.com Sec-WebSocket-Protocol: chat, superchat Sec-WebSocket-Version: 13 The handshake from the server looks as follows: HTTP/1.1 101 Switching Protocols Upgrade: websocket Connection: Upgrade Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo= Sec-WebSocket-Protocol: chat The leading line from the client follows the Request-Line format. The leading line from the server follows the Status-Line format. The Request-Line and Status-Line productions are defined in [RFC2616]. https://www.rfc-editor.org/rfc/rfc6455

	WebSocket connections are established by *upgrading* an HTTP request/response pair. A client that supports WebSockets and wants to establish a connection will send an HTTP request that includes a few required headers: • `Connection: Upgrade` ○ The `Connection` header generally controls whether or not the network connection stays open after the current transaction finishes. A common value for this header is `keep-alive` to make sure the connection is persistent to allow for subsequent requests to the same server. During the WebSocket opening handshake we set to header to `Upgrade` , signaling that we want to keep the connection alive, and use it for non-HTTP requests. • `Upgrade: websocket` ○ The `Upgrade` header is used by clients to ask the server to switch to one of the listed protocols, in descending preference order. We specify `websocket` here to signal that the client wants to establish a WebSocket connection. https://sookocheff.com/post/networking/how-do-websockets-work/ Once a client sends the initial request to open a WebSocket connection, it waits for the server's reply. The reply must have an `HTTP 101 Switching Protocols` response code. The `HTTP 101 Switching Protocols` response indicates that the server is switching to the protocol that the client requested in its `Upgrade` request header. In addition, the server must include HTTP headers that validate the connection was successfully upgraded: After the client receives the server response, the WebSocket connection is open to start transmitting data. https://sookocheff.com/post/networking/how-do-websockets-work/

	How Do Websockets Work? WebSockets, on the other hand, allow for sending message-based data, similar to UDP, but with the reliability of TCP. WebSocket uses HTTP as the initial transport mechanism, but keeps the TCP connection alive after the HTTP response is received so that it can be used for sending messages between client and server. WebSockets allow us to build "real-time" applications without the use of long-polling. WebSockets begin life as a standard HTTP request and response. Within that request response chain, the client asks to open a WebSocket connection, and the server responds (if its able to). If this initial handshake is successful, the client and server have agreed to use the existing TCP/IP connection that was established for the HTTP request as a WebSocket connection. Data can now flow over this connection using a basic framed message protocol. Once both parties acknowledge that the WebSocket connection should be closed, the TCP connection is torn down. https://sookocheff.com/post/networking/how-do-websockets-work/
negotiating between said first node and said second node a transactional role reversal, said step of negotiating referencing said original network connection; and	The accused product practices negotiating between said first node (e.g., client) and said second node (e.g., server) a transactional role reversal, said step of negotiating referencing said original network connection (e.g., TCP connection). WebSocket connections are established by upgrading an HTTP request/response pair. A client that supports WebSocket and wants to establish a connection will send an HTTP request to the server. Once a client sends the initial request to open a WebSocket connection, it waits for the server's reply. After the client receives an affirmative server response, the WebSocket connection is opened to start

transmitting data. In WebSocket communication, both sides i.e., the client and the server can send data at will. That is, the server need not be restricted to only relaying responses and the client only sending requests. Thus, the first and second network nodes negotiate transactional role reversal. During negotiation, existing TCP/IP connection is maintained.

1.2. Protocol Overview

This section is non-normative.

The protocol has two parts: a handshake and the data transfer.

The handshake from the client looks as follows:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

The handshake from the server looks as follows:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

The leading line from the client follows the Request-Line format. The leading line from the server follows the Status-Line format. The Request-Line and Status-Line productions are defined in [RFC2616].

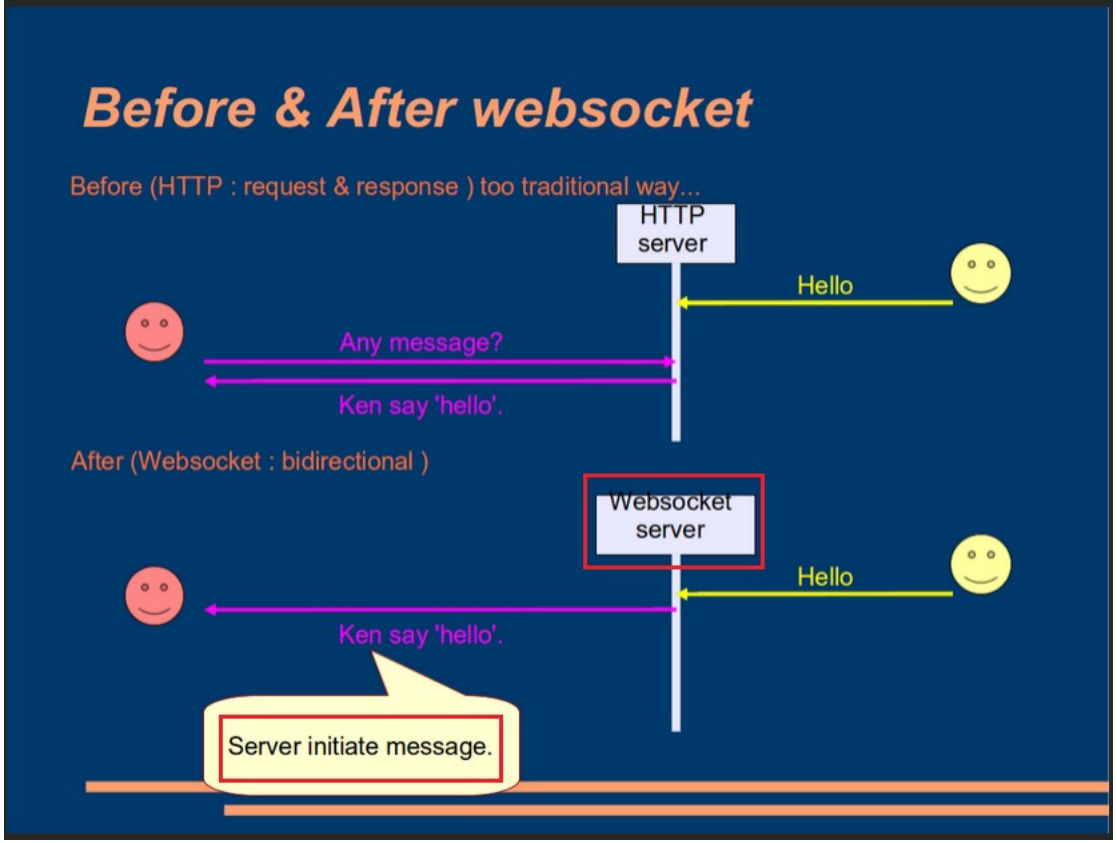
https://www.rfc-editor.org/rfc/rfc6455

	`Once the client and server have both sent their handshakes, and if the handshake was successful, then the data transfer part starts. This is a two-way communication channel where each side can, independently from the other, send data at will.` https://www.rfc-editor.org/rfc/rfc6455 WebSocket connections are established by *upgrading* an HTTP request/response pair. A client that supports WebSockets and wants to establish a connection will send an HTTP request that includes a few required headers: • `Connection: Upgrade` ◦ The `Connection` header generally controls whether or not the network connection stays open after the current transaction finishes. A common value for this header is `keep-alive` to make sure the connection is persistent to allow for subsequent requests to the same server. During the WebSocket opening handshake we set to header to `Upgrade`, signaling that we want to keep the connection alive, and use it for non-HTTP requests. • `Upgrade: websocket` ◦ The `Upgrade` header is used by clients to ask the server to switch to one of the listed protocols, in descending preference order. We specify `websocket` here to signal that the client wants to establish a WebSocket connection. https://sookocheff.com/post/networking/how-do-websockets-work/

	Once a client sends the initial request to open a WebSocket connection, it waits for the server's reply. The reply must have an `HTTP 101 Switching Protocols` response code. The `HTTP 101 Switching Protocols` response indicates that the server is switching to the protocol that the client requested in its `Upgrade` request header. In addition, the server must include HTTP headers that validate the connection was successfully upgraded: After the client receives the server response, the WebSocket connection is open to start transmitting data. https://sookocheff.com/post/networking/how-do-websockets-work/ **How Do Websockets Work?** WebSockets, on the other hand, allow for sending message-based data, similar to UDP, but with the reliability of TCP. WebSocket uses HTTP as the initial transport mechanism, but keeps the TCP connection alive after the HTTP response is received so that it can be used for sending messages between client and server. WebSockets allow us to build "real-time" applications without the use of long-polling. https://sookocheff.com/post/networking/how-do-websockets-work/
establishing a second asymmetric transactional session, said session being uniquely identifiable by said URI, said first node enacting said second transactional role and said second	The accused product practices establishing a second asymmetric transactional session (e.g., session for sending data from server to client), said session being uniquely identifiable by said URI, said first node (e.g., client) enacting said second transactional role (e.g., server) and said second node (e.g., server) enacting said first transactional role (e.g., client). WebSocket connections are established by upgrading an HTTP request/response pair. A client that supports WebSocket and wants to establish a connection will send an HTTP request. Once a client sends the initial request to open a WebSocket connection, it waits for the server's reply. The reply must have an HTTP 101

node enacting said first transactional role,	Switching Protocols response code. The HTTP 101 Switching Protocols response indicates that the server is switching to the protocol that the client requested in its Upgrade request header. After the client receives the server response, the WebSocket connection is open to start transmitting data. During WebSocket connection, both the server and the client can relay data independent of each other. Here, a server can push information to the client and of course, the client can respond. Thus, it can be considered that the first and second nodes establishes a second asymmetric transactional session, where each node can enact the initial transactional role of the other. During the handshake, a Request-URI ("uniquely identifiable session") is used to identify the endpoint of the WebSocket connection. Thus, the session is uniquely identifiable. **1.2. Protocol Overview** _This section is non-normative._ The protocol has two parts: a handshake and the data transfer. The handshake from the client looks as follows: `GET /chat HTTP/1.1` `Host: server.example.com` `Upgrade: websocket` `Connection: Upgrade` `Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==` `Origin: http://example.com` `Sec-WebSocket-Protocol: chat, superchat` `Sec-WebSocket-Version: 13` The handshake from the server looks as follows: `HTTP/1.1 101 Switching Protocols` `Upgrade: websocket` `Connection: Upgrade` `Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=` `Sec-WebSocket-Protocol: chat`

	https://www.rfc-editor.org/rfc/rfc6455 Once the client and server have both sent their handshakes, and if the handshake was successful, then the data transfer part starts. This is a two-way communication channel where each side can, independently from the other, send data at will. https://www.rfc-editor.org/rfc/rfc6455 WebSocket solves a few issues with HTTP: • Bi-directional protocol — either client/server can send a message to the other party (In HTTP, the request is always initiated by the client and the response is processed by the server — making HTTP a uni-directional protocol) • Full-duplex communication — client and server can talk to each other independently at the same time. • Single TCP connection — After upgrading the HTTP connection in the beginning, client and server communicate over that same TCP connection throughout the lifecycle of WebSocket connection. https://medium.com/platform-engineer/web-api-design-35df8167460

	• WebSocket is a protocol providing full-duplex communication channels over a single TCP connection. Where as, HTTP providing half-duplex communication. • Information exchange mode of WebSocket is bidirectional. Means, server can push information to the client (which does not allow direct HTTP).  https://developerinsider.co/difference-between-http-and-http-2-0-websocket/

	GET /chat HTTP/1.1 Host: server.example.com Upgrade: websocket Connection: Upgrade Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== Origin: http://example.com Sec-WebSocket-Protocol: chat, superchat Sec-WebSocket-Version: 13 In compliance with [RFC2616], header fields in the handshake may be sent by the client in any order, so the order in which different header fields are received is not significant. The "Request-URI" of the GET method [RFC2616] is used to identify the endpoint of the WebSocket connection, both to allow multiple domains to be served from one IP address and to allow multiple WebSocket endpoints to be served by a single server. The client includes the hostname in the |Host| header field of its handshake as per [RFC2616], so that both the client and the server can verify that they agree on which host is in use. https://www.rfc-editor.org/rfc/rfc6455
wherein said uniquely identifiable session uses a network connection traversing hardware enforcing asymmetric communication.	The accused product practices - wherein said uniquely identifiable session (e.g. Request-URI) uses a network connection traversing hardware enforcing asymmetric communication. The WebSocket is designed to work over HTTP ports 80 and 443. The opening handshake between the server and the client is compatible with HTTP-based server-side software and intermediaries. During the handshake, a Request-URI is used to identify the endpoint of the WebSocket connection. Thus, it can be said that the uniquely identifiable session uses a network connection traversing hardware enforcing asymmetric communication.

	The WebSocket Protocol is designed to supersede existing bidirectional communication technologies that use HTTP as a transport layer to benefit from existing infrastructure (proxies, filtering, authentication). Such technologies were implemented as trade-offs between efficiency and reliability because HTTP was not initially meant to be used for bidirectional communication (see [RFC6202] for further discussion). The WebSocket Protocol attempts to address the goals of existing bidirectional HTTP technologies in the context of the existing HTTP infrastructure; as such, it is designed to work over HTTP ports 80 and 443 as well as to support HTTP proxies and intermediaries, even if this implies some complexity specific to the current environment. However, the design does not limit WebSocket to HTTP, and future implementations could use a simpler handshake over a https://www.rfc-editor.org/rfc/rfc6455

1.3. Opening Handshake

This section is non-normative.

The opening handshake is intended to be compatible with HTTP-based server-side software and intermediaries, so that a single port can be used by both HTTP clients talking to that server and WebSocket clients talking to that server. To this end, the WebSocket client's handshake is an HTTP Upgrade request:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

In compliance with [RFC2616], header fields in the handshake may be sent by the client in any order, so the order in which different header fields are received is not significant.

The "Request-URI" of the GET method [RFC2616] is used to identify the endpoint of the WebSocket connection, both to allow multiple domains to be served from one IP address and to allow multiple WebSocket endpoints to be served by a single server.

The client includes the hostname in the |Host| header field of its handshake as per [RFC2616], so that both the client and the server can verify that they agree on which host is in use.

https://www.rfc-editor.org/rfc/rfc6455